Refining binned aggregation of unevenly spaced, real-time data for visualisation and control

Kirsten du Toit
CSIR, South Africa

Thomas Fogwill
CSIR, South Africa

Tami Maiwashe
CSIR, South Africa

Melanie van der Westhuizen
CSIR, South Africa

Abstract
The resource constrained environment in South Africa has led to an increased interest in the monitoring and control of electricity demand. The associated proliferation of smart meters/sensors and the collection of their data allows for monitoring and control of demand on a much wider scale than previously possible. A homeowner, business or utility may now make use of real-time data to track and control their electricity demand with potentially widespread economic effect. We examine the characteristics that a system must exhibit to excel at high volume, low latency real-time data stream processing to support such applications. A robust, scalable and fast method for real-time processing of streamed electricity demand data is suggested. This method also addresses the problems associated with developing countries in that the data received are sometimes inconsistent, unevenly spaced and interrupted. Processing these unevenly spaced data for visualisation is an important requirement in order for users to more easily understand and analyse their usage trends. Binned aggregation is explored as an effective strategy for minimizing the negative effects of inconsistent data of different frequencies. It is also used to reduce the data to manageable dimension, while real-time processing determines where and how to use the data. A sensor's data are stored in several rolling arrays of differing size that represent different time-frames. We keep the most detailed information for the current time-frame. The format chosen allows for data reduction yet allows redundant data to contribute to the model and refine the aggregation. The reduced amount of data stored allows fast retrieval of those data for visual display and analysis and improved scalability.

Keywords
Real-time stream processing, unevenly spaced data, binned aggregation, visualisation, time-series, sensor data

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

Introduction

As smart meters become more popular, it becomes theoretically possible to monitor and control energy demand by aggregating the time-series sensor data collected from these meters. Sensor data are typified by a time-stamp and a value. However, continuous long-term recording results in huge data sets that poses a challenge to storage and analysis of the data. These data-sets need to be reduced to a format that allows for fast queries for retrieval of data for visualisation and analysis. We want to store as little data as possible and in the most meaningful format.

Depending on the network availability and speed, the sensor data may come through unevenly and inconsistently, in bursts and sometimes out of sequence. The system we propose must be able to deal with this problem.

Monitoring and control also entail a real-time component. If an immediate response is not required then one can store all the incoming data in a large data warehouse and run historical queries on them in order to obtain information of interest. However, if the software is required to control devices in near-real-time, it needs to be able to react as quickly as possible and raise an alarm or take action if demand is too high. The processing engine thus needs to adopt a model of inbound processing, where processing is performed directly and immediately on incoming data before (or instead of) storing them. It also needs to be able to deal with high volumes of data and to be able to automatically scale in terms of load and processing power.

Monitoring requires current data. The older the data, the less important they are for monitoring and control purposes. The data of highest importance are the most current data and we propose a scheme that stores data in lowering resolutions for different time-frames depending on the age of the data.

Table 1 summarises the system requirements for real-time data processing for visualisation and control.

System Requirement	Description	Reference
Scalable	Cope with high volumes of sensor data and allow for parallel processing	Balakrishnan, H., Balazinska, M., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., ... & Zdonik, S. (2004).
Robust	Deal with potentially unevenly spaced data, and data that are received out of sequence. Deal with bursts and interruptions in data.	Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). Balakrishnan et al (2004)
Data reduction for visualisation	Store data in a format suited for visual analysis. Allow for data reduction, reduce large sets of data to small enough sets for visual display and analysis, without losing data of interest (i.e. the maximums, minimums, means and frequency).Allow redundant data to refine the model immediately without cluttering by storing it unnecessarily	Liu, Z., Jiang, B., & Heer, J. (2013, June) Aris, A., Shneiderman, B., Plaisant, C., Shmueli, G., & Jank, W. (2005)
Fast	Low latency, immediate processing - Process the data with the purpose of monitoring and control - process these data before storing them in order to immediately raise alarms or take action (for example control demand by switching off electricity, i.e. load shedding).	Stonebraker et al (2005) Balakrishnan et al (2004)

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

Data reduction for monitoring	Keep the highest resolution of aggregated data for the current time-frame (which is of highest interest) - accurate monitoring and control require current information	Hao, M. C., Dayal, U., Keim, D. A., & Schreck, T. (2007)

Table 1: Real-time data processing for visualization and control requirements

Related literature

Reducing time-series data has been fairly extensively researched, but we require reducing the data for visualisation and for monitoring. Hao et al (2007) observe that the less current an observation is, the less important it is for the monitoring task and that newest data can be assumed to be of the most interest.

In their paper discussing methods for interactive visualisation, Liu et al (2013) follow the principle that data should be reduced according to the chosen resolution of the visualized data, not depending on the number of records. In support of this principle, they review several data reduction methods, including filtering, sampling and aggregation and then select binned aggregation as their primary data reduction strategy because it conveys both global patterns (e.g. densities) and local features (e.g. outliers), while enabling multiple levels of resolution via the choice of bin size.

The data we receive are not evenly spaced and some work has also been done on representing unevenly spaced time-series data. Aris et al (2005) compare 4 methods namely: sampled events, aggregated sampled events, event index and interleaved event index. They maintain that aggregation can be used to allow the overview to fit into the screen, decrease the amount of processing required and maintain a high level of responsiveness. They observe that as the aggregation level increases, the precision degrades and declare that one possibility is to aggregate dynamically considering the amount of information to be displayed and available space on the screen.
Although we are more interested in current data and an immediate response to data received, we examined the work of Keller, M., Beutel, J., Saukh, O., & Thiele, L. (2012, October) which deals with historical queries on large sensor data-sets. They discuss methods for visualising large sensor data-sets and present a middle-ware design that significantly reduces request response time by the use of caching techniques and pre-computed data. Scalability, immediate visualisation and analysis is important to us however and we favour reducing our data over improved methods of querying large historical data-sets.

Storing the data in a format suited for visualisation and analysis is critical. Goldstein, R., Glueck, M., & Khan, A. (2011, November) presented work on time-series data and maintain that a big issue is that relevant time scales can vary by orders of magnitude depending on the desired analysis. They suggest that lossy compression can be applied in real-time as data are acquired to ensure that a smaller set of a time-series data is available when needed at an appropriate resolution, They propose an algorithm that compresses a piecewise constant time series by merging segments within a sliding time window, a procedure which preserves prominent edges and spikes.

Their method involves the application of a compression buffer into which newly acquired data-points are added in real time while limiting the size and the time-frame of the buffer. They discuss different ways of producing different resolutions of data and decide to apply

multiple compression buffers directly to the original time-series with different values of time resolution (time-frame / bin size). Their paper merges work on visualisation and real-time data reduction for visualisation and analysis as well as the exciting suggestion that the time-series be stored in multiple compression buffers of varying bin size.

Continuously processing sensor data is an intensive process and the characteristics that a real-time data processing engine should exhibit are described in fair detail by Stonebraker et al (2005). They discuss how data should be kept moving and that in order to achieve low latency, a system must be able to perform message processing without having a costly storage operation in the critical processing path. They suggest the use of an active system using event/data driven processing in order to avoid the latency incurred by polling passive systems.They also suggest that SQL should be used to process the moving real-time streams of data because relying on low-level programming languages results in long development cycles and high maintenance costs.

According to Stonebraker et al (2005), the stream processing engine should be robust and provide resiliency against stream “imperfections”, including missing data and data that are received out-of-order. Scalability is very important and they conclude that a stream processing application must have a highly-optimized, minimal-overhead execution engine to deliver real-time response for high-volume applications.

Further work on stream processing engines has been published by Balakrishnan et al (2004). They maintain that stream processing engines should support time series operations, which are not well supported by current DBMSs. They agree that the engine should respond immediately to real-time data and perform processing directly on incoming messages before (or instead of) storing them.

Our contribution – a binned aggregation based strategy

We combine parts of most of the above ideas, in order to produce a scalable system capable of dealing with high volumes of inconsistent data, while providing fast response for control and monitoring and storing the data in a format ideal for visual monitoring.
We will reduce data using binned aggregation and make use of the idea of dynamic real-time aggregation in order to minimise latency. We also make use of the idea of storing data in several different buffers of predetermined varying bin size. i.e. one second, one minute, one hour, one day, one month. This will also negate the need for specific time series operators that are otherwise required to reduce query time, given that DBMS's do not deal well with time operator queries. Retrieving and processing historical data will be much faster because they are ordered and stored in indexes that have a predefined associated time-stamp.

We make use of the idea of a sliding time window in that we store our reduced aggregated data in rolling arrays. These rolling arrays are fixed width arrays that comprise bins that hold aggregated data for a specific time period – e.g. the last hour. We do not shift data within that array. Instead, we use indexes to indicate the virtual start and end of the rolling array. Data for the next hour roll onto and overwrite the data for the previous hour. Using a number of sliding time windows also allows us to keep data in arrays that decrease in resolution the older the data are. The most detailed data will always be kept for the time of

most interest in terms of monitoring and visualization, i.e. the current hour.
We make sure that our system is event/data-driven in order to minimize latency, and perform processing immediately on the data before storage.

Proposed method

Designing for speed and scalability

Data will be processed and required actions taken before storage in order to minimise latency and reaction time. We will use Node.js which allows asynchronous parallel processing by allowing multiple processes to handle the streams. This will make the system highly scalable and allow it to deal equally well with low and high volumes of data.
We store data in mongoDB which allows for high availability, high scalability and allows batch data processing and aggregate calculations using native mongoDB operations.

Rolling array format

In order to minimise database writes and thereby increase speed, we choose to store data in rolling arrays and do not shift the data within the array. The virtual start of the array is the array index at which we store the oldest data we still track in this time period, while the virtual end index is where we store the newest data. The current time determines the virtual end of the array, and thus also the virtual start. For instance, in a rolling array of per minute data for the last hour, the array will be of length 60 (one bin for each minute in the hour). If the current time is 10:53, then the virtual end of the array is the 53rd bin (at index 52), and the virtual start is at index 53. If we receive data for 10:05, we aggregate and write it to the bin at index 04. When we get to the end of the actual array (e.g. beyond index 59), we “wrap around” (by taking index modulo 60, in this example) to the start of the array. This reuses the slots at the beginning and overwrites old data that we no longer wish to track. In this way, the array always keeps the data for a fixed time period (1 hour in our example) and any time within that time period can be converted simply into the index of the bin holding the data for that time.

Figure 1 illustrates how data would be stored in an array that represents one week with a bin size of one day, starting with a Monday. When data is received on Saturday it is written into index 5, Sunday's data is written into index 6, and Monday's data is written to Index 0.

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

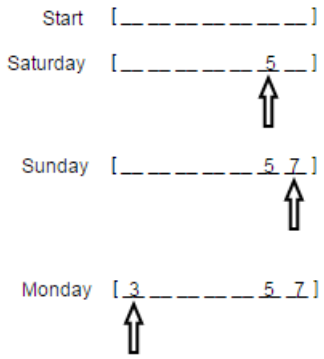

Figure 1: Aggregating data into a rolling array

Reducing data for visualisation

In the context of control and monitoring, the most important time-frame is the current one. Data will be reduced by binned aggregation into 5 differently sized rolling arrays each each representing a different time-frame, while keeping the most data for the current hour.

- ONE HOUR: An array for the current hour comprised of 3600 seconds. i.e. the array consists of 3600 bins, each representing 1 second in the last hour. The bin at index 0 in the array will hold all the aggregated data generated by the sensor during the first second of the hour. The bin size for the aggregation for this array will be one second. This means that all data from the sensor received with a time-stamp that is within that one second period will be processed for the index associated with that period.
- ONE DAY: An array for the current day comprised of 1440 minutes. The fixed time slice size for this array (also known as the bin size) will be one minute. i.e. all data received in the minute from 08:00 to 08:01 will be aggregated and held in index 480 which represents all data received during the 480th minute of the day.
- ONE WEEK: An array for the current week comprised of one hour bins. This array holds 168 values.
- ONE MONTH: An array for the current month comprised of one day bins. This array holds 31 values.
- ONE YEAR: An array for the current year comprised of one month bins. This array holds 12 values.

The normal screen size is 1024 pixels wide so even if we employ the entire width for plotting the hour's data, it may not be enough. However data are rarely received every second for a sensor and as an example, if we receive data every 5 seconds, we will only need to plot 720 points. The graphing utility we use allows zooming and panning and extra detail is not lost but still available for visual display.

For each bin/period within each time-frame we will store a number of different values. We will keep the maximum value of all data received from that sensor during that time slice. We will also store the minimum value received, the sum of all values received and the number of times values are received (frequency) for that period from that sensor.
As an example, assuming we receive data every 6 seconds from a sensor, we will receive 10 values in a minute. At the end of the first hour the one hour rolling array will hold 600

out of 3600 values, with the bins that data were not received for holding nulls. The one day time-frame array, however, will hold a value for each minute, because at least one value is generated and received each minute.

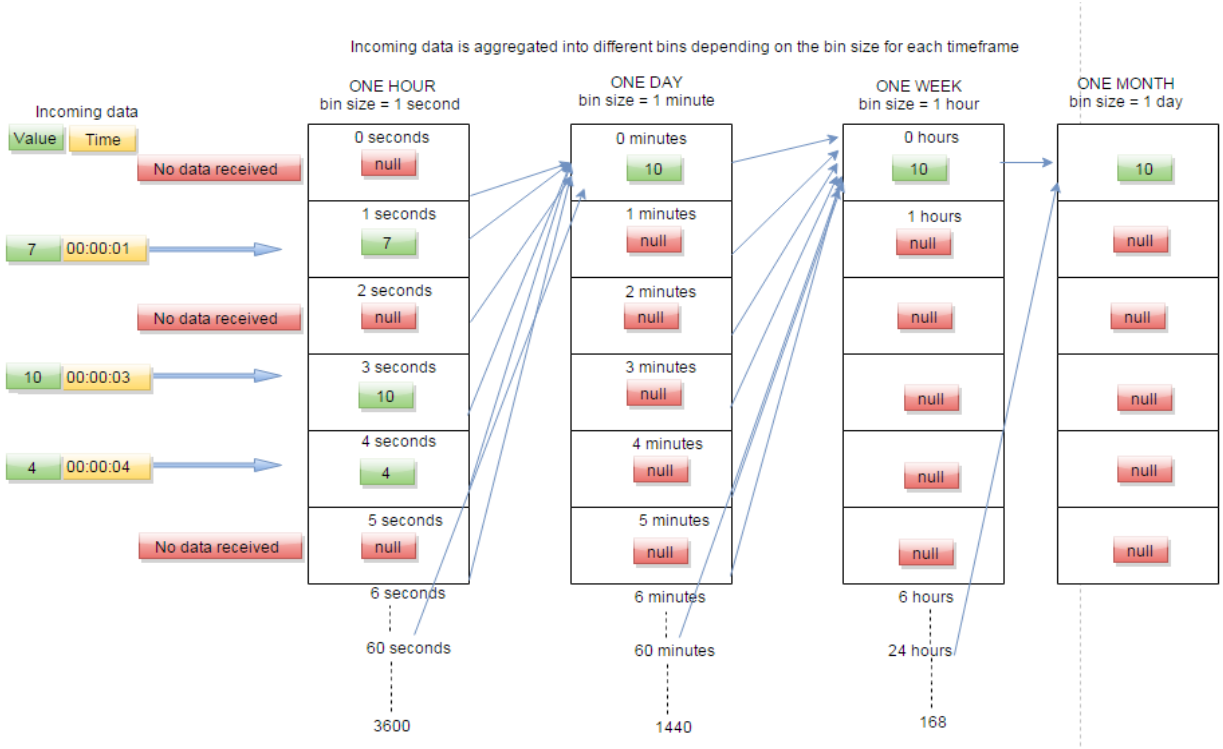

Figure 2: Aggregating data into different rolling time-frame arrays.

From Figure 2 we can see that all data generated within the first minute of the day are stored in bins representing one second in the ONE HOUR time-frame. All data generated within the first minute are aggregated into the first bin of the ONE DAY time-frame. Similarly all data from the first 60 minutes within the ONE DAY time-frame are aggregated into the first bin of the ONE MONTH time-frame.

Figure 3: The data format per time-frame.

Robustness

For each sensor we also keep the time-stamp of the last received data the last updated index for each array. As data enter the system, their indexes for each of the arrays are calculated, and they are aggregated into the relevant bin. If data arrive and hour late, they will not be aggregated into the hour time-frame array because that array only keeps data for the last hour, however they will be aggregated into the current day, week, month and year arrays. If 10 values are received in the same period and they do not affect the maximum and minimum arrays, they will still refine the model by being added into the sum and incrementing the frequency for that period. This means that no extra space is used to store data, but that data are not lost. The total and frequency are used to refine the mean for that period.

In order not to lose data of interest (i.e. peaks, minimums, and means) and to ensure that aggregates can be recomputed efficiently if data arrive late, data for each time-frame are aggregated and stored over 4 arrays. Figure 3 demonstrates these arrays; one for the maximums, one for the minimums, one for the totals and one for the frequencies of data generated during the time slice associated with that bin. As new data arrive, these are sufficient to recalculate new aggregates. For electricity demand monitoring, the maximum (or peak) demand is the most interesting aggregate to track.

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

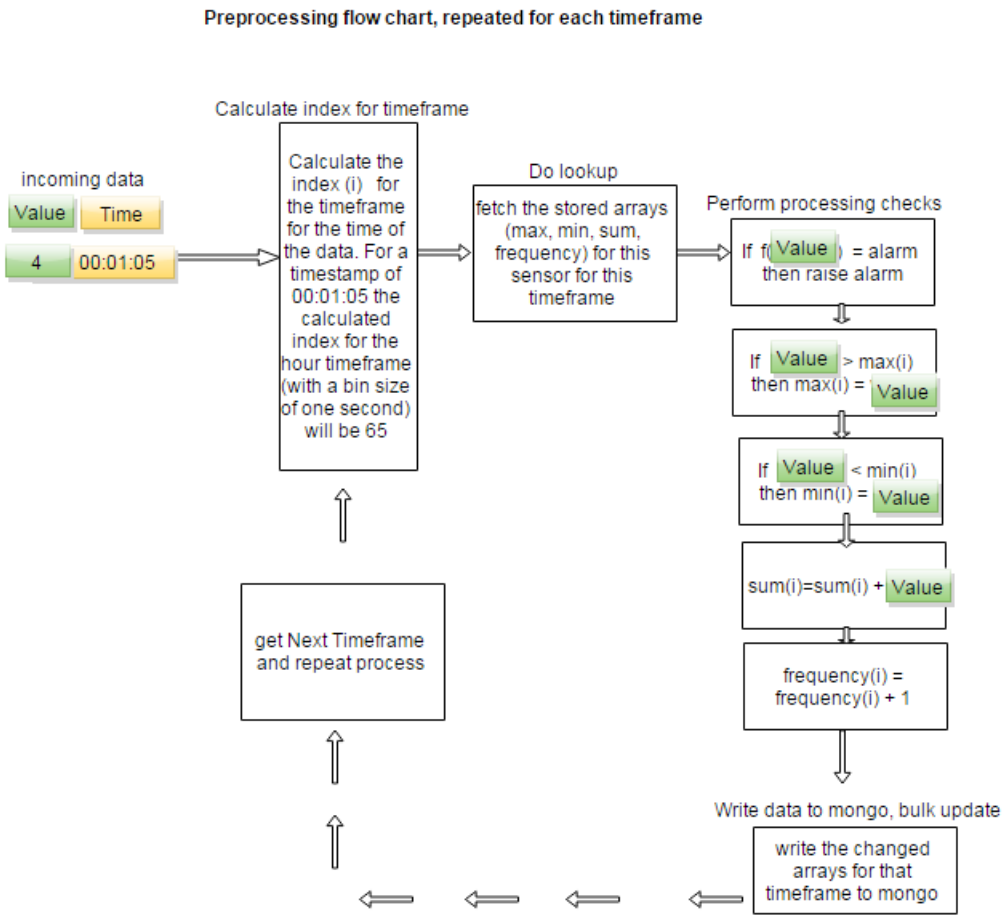

Figure 4: Processing the data

Immediate Processing

Figure 4 illustrates the processing of data received. It can be seen that the data are immediately reacted upon, before storage.

The processing is data driven, which means that the system responds immediately to received data. As soon as a value is received we aggregate it into the array for that sensor for the first time-frame (ONE HOUR). Starting with this time-frame array (3600 values = one hour) we first calculate the index of the array that this value should be aggregated into and do a look-up of stored data for this sensor. The index it should be aggregated into depends on the time-stamp of the value which tells us when the value was generated, not when it was received. Alarm functions are defined for the value and it can immediately be calculated whether the new value is cause for an alarm to be raised or action to be taken (i.e. load shedding). We then perform the processing which involves deciding whether the index for the new value is an entirely new index or has already been used to store data for this period. If the time-stamp for the value falls into an entirely new period then we need to deal with inconsistent unevenly spaced data by blanking all the bins that fall between the last value received and the new value received. We then aggregate the value received into

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

the correct bin for the 4 different arrays we use to keep track of maximums, minimums, sums and frequencies of data received.

Assuming the calculated index for the time-stamp of the value received is *i*, and the value of the value received is *newvalue*, the following rules calculate the aggregates.

If *newvalue* > max[i] (the maximum value received for all values for that period) then max[i] = *newvalue*.

If *newvalue* < min[i] (the minimum value received for all values for that period) then min[i] = *newvalue*.

We always add the value to the sum:

sum[i] = sum[i] + *newvalue*.

And always increment the frequency.

frequency[i] = frequency[i] + 1.

We then write the changed data to the mongoDB in an efficient, asynchronous batch update and begin the whole process again for the same value but for the next time-frame, i.e. the day time-frame which contains aggregated data for each minute.

Time-series operations

We negate the need for using a database that supports fast time-series operators by processing and storing our data in time referenced arrays. Each bin represents the data generated during a fixed time slice size that is the bin size for that array. This means that time-series operations are greatly simplified.

Inconsistent unevenly spaced data

Because the data cannot be relied upon to be evenly spaced and consistent, data may not be generated to fill every bin, i.e. we may not receive data every second in order to fill our hour array of 3600 seconds. Our processing is data driven and because we use rolling arrays to store sliding windows of data it is thus important to keep track of the time-stamp of our last received data and to use this in conjunction with the time-stamp of our newly received data in order to overwrite or blank old data for bins for which we have not received new data.

Concurrency and synchronization complications

In essence because we are using the stored latest time-stamp to calculate whether to blank intermediate bins or not, we are keeping state, and this leads to concurrency problems when multiple processes use the same variables for state. In order to deal with this problem the mongoDB collections for that sensor are locked while updating the relevant arrays and latest time-stamp variable. In addition, updates to the arrays are done in batch, to prevent inconsistencies from data arriving out of order.

Resulting visualisations

The above design meets the requirements as laid out previously. Testing our design resulted in the following visualisations: It is important to note that this test involved data generated with a frequency of approximately once every ten minutes.

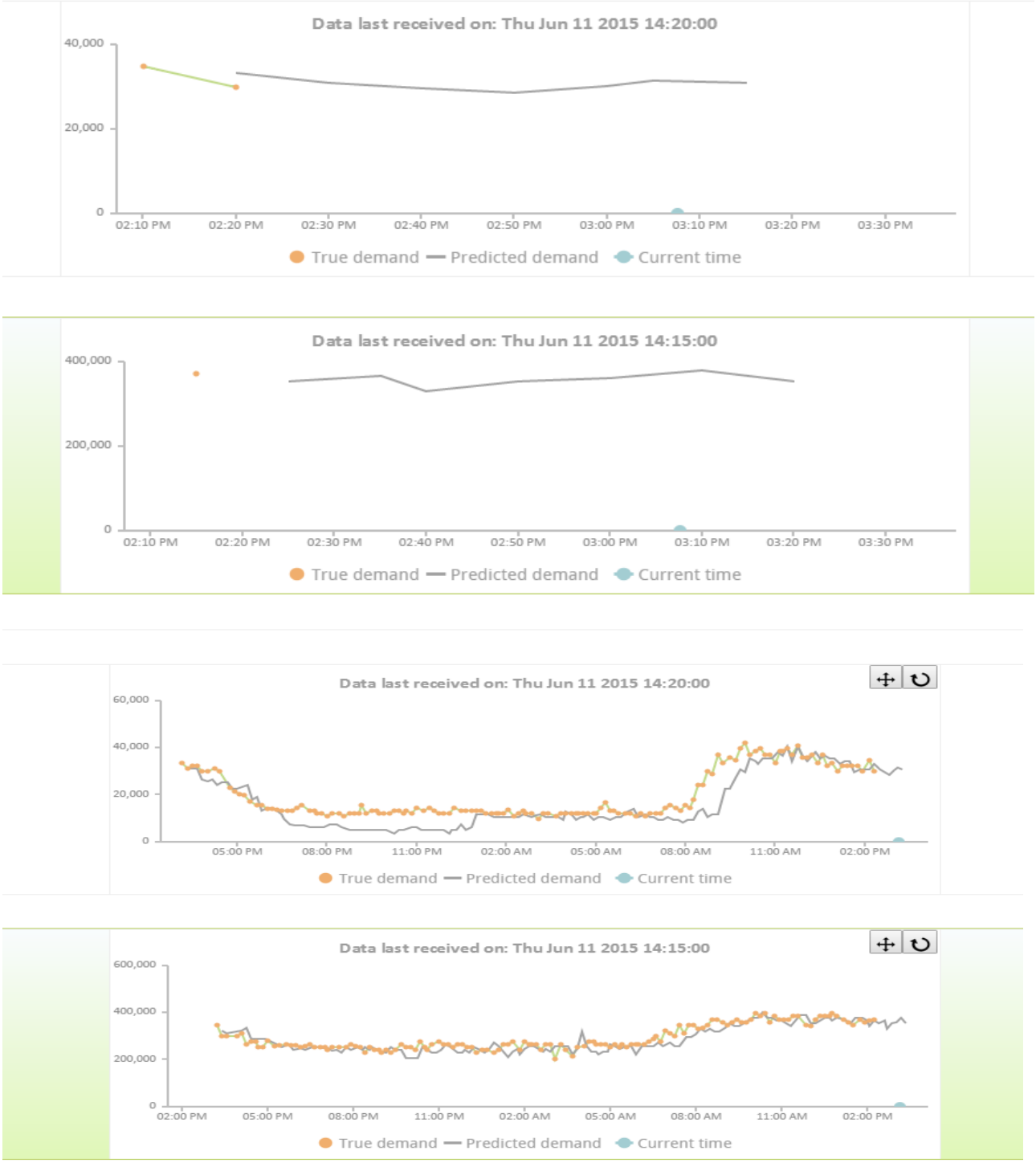

Figure 5: The ONE HOUR time-frame for 2 different sensors

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

Figure 5 illustrates the aggregated data received from 2 different sensors. These data were generated every ten to fifteen minutes. It is obvious from the above graphs that not enough data were received to make sense of the ONE HOUR time-frame.

Figure 6: The ONE DAY time-frame

Figure 6 illustrates the data aggregated from those same two sensors into the ONE DAY time-frame which stores data in one minute sized bins. This view of the data is much easier to analyse, allowing us to see that electricity demand increases from approximately 8 a.m.

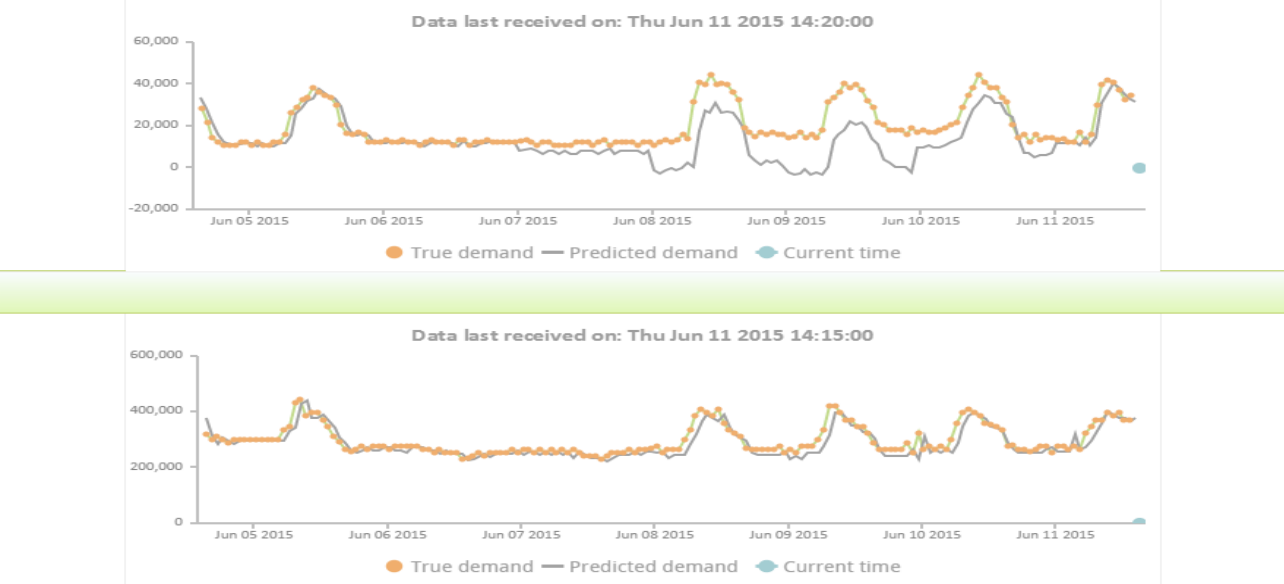

Figure 7: The ONE WEEK time-frame

Figure 7 shows us data from the same sensors that have been aggregated into the one hour bins for the ONE WEEK time-frame. This is the best graph for analysis. Data have been generated and received at least once per bin and this results in the best detailed overview given the frequency of the data received. It is possible to compare week day demand.

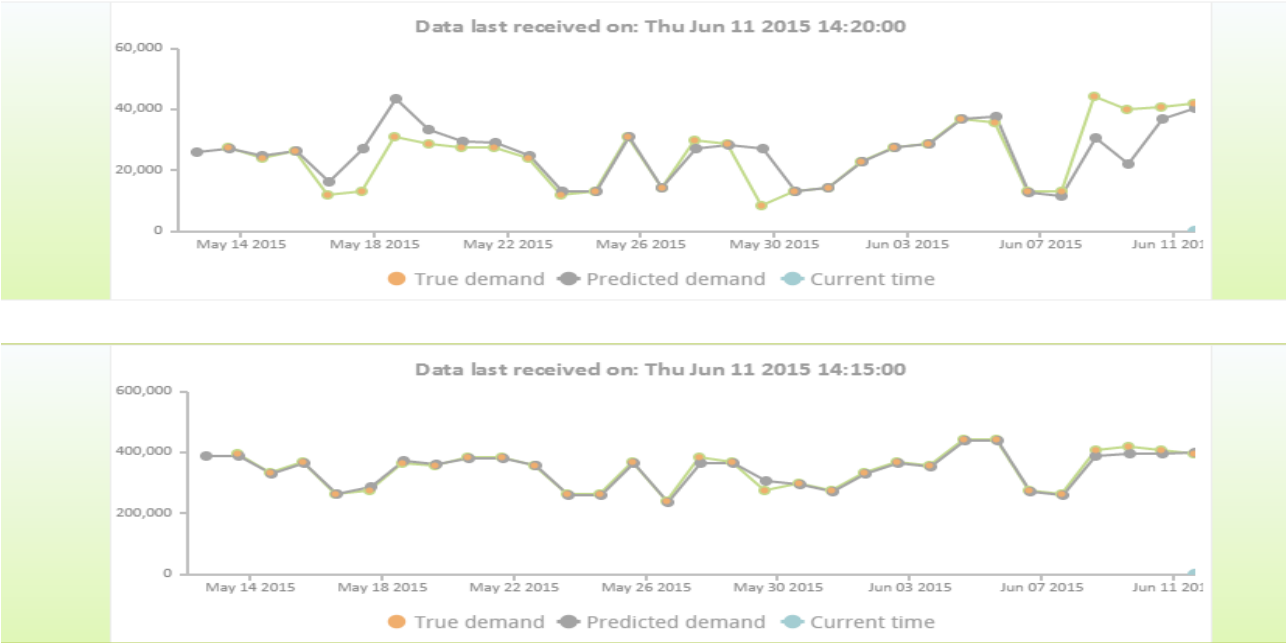

Figure 8: The ONE MONTH time-frame

Figure 8 shows us data from the same sensors that have been aggregated into the one day bins for the ONE MONTH time-frame. The graph allows us to compare days within a month, noting that the least demand is consistently recorded over weekends.

Conclusions and future work

We optimise speed of response by processing the data as we receive them. We reduce the data using binned aggregation into multiple arrays each representing a different sized sliding window in time and we do this in decreasing detail, laying the most emphasis on the most recent data.

Our data aggregation process deals well with different frequencies of data received. Even though there were not enough data to make sense of the ONE HOUR time-frame, there were more than enough data to fill the ONE DAY time-frame and allow for analysis and monitoring. One can clearly see that power consumption increases during daylight hours. The ONE WEEK time-frame graph allows one to see the increased consumption during weekdays and compare on a day by day basis. The ONE MONTH graph allows one to compare weekdays and weekend days within a month.

The most use possible is made of whatever data we receive at whatever frequency they are generated. The data are split over differently sized bins that allow us wider scales of comparison.

Initial load testing on an average sized computer with a single Node.js process and single mongoDB instance, sees the process handling approximately 20 values per second. Loading the single node process beyond that leads to rapid deterioration of performance. However Node.js scales well laterally, and running a second Node.js process in parallel almost doubles the number of values handled per second. In the future we would like to compare more data of varying frequencies and properly test the scalability and

Steyn, J., Van Belle, J.P. (Eds.). 2015. Beyond development. Time for a new ICT4D paradigm?
Proceedings of the 9th IDIA conference, IDIA2015, Nungwi, Zanzibar: 163-176
ISBN: 978-0-620-68395-1

responsiveness of the design as well as examine aggregating data by location.

System Requirement	Description	Requirement met
Scalable	Node.js allows for parallel asynchronous processing, mongoDB provides high availability and scalability as well as batch aggregation update capabilities.	yes
Robust	Has no problem dealing with out of sequence data and different frequencies of data.	yes
Data reduction for visualisation	Reduces data into intuitive bin sizes which allow for visualisation and analysis of data during different time-frames and minimizes the problems associated with unevenly spaced data (data of different frequencies)	yes
Fast	Immediate inbound processing.	yes
Data reduction for monitoring	Keep data in decreasing resolutions, with the most detail kept for the most recent data	yes

Table 2: Design criteria summary – adapted from Table 1

References

Aris, A., Shneiderman, B., Plaisant, C., Shmueli, G., & Jank, W. (2005). Representing unevenly-spaced time series data for visualization and interactive exploration.
In *Human-Computer Interaction-INTERACT 2005* (pp. 835-846). Springer Berlin Heidelberg.

Balakrishnan, H., Balazinska, M., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., ... & Zdonik, S. (2004). Retrospective on aurora.
The VLDB Journal, *13*(4), 370-383.

Goldstein, R., Glueck, M., & Khan, A. (2011, November). Real-Time Compression of Time Series Building Performance Data.
In *Proceedings of Building Simulation*.

Hao, M. C., Dayal, U., Keim, D. A., & Schreck, T. (2007). Multi-resolution techniques for visual exploration of large time-series data.

Keller, M., Beutel, J., Saukh, O., & Thiele, L. (2012, October). Visualizing large sensor network data sets in space and time with vizzly.
In *Local Computer Networks Workshops (LCN Workshops), 2012 IEEE 37th Conference on* (pp. 925-933). IEEE.

Liu, Z., Jiang, B., & Heer, J. (2013, June). imMens: Real‐time Visual Querying of Big Data.
In *Computer Graphics Forum* (Vol. 32, No. 3pt4, pp. 421-430). Blackwell Publishing Ltd.

Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005). The 8 requirements of real-time stream processing.
ACM SIGMOD Record, *34*(4), 42-47.